

Code Coverage

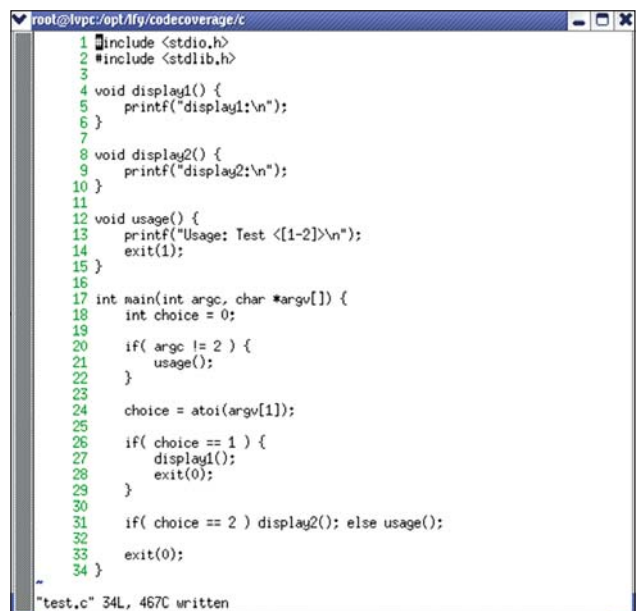
Delivering high quality, defect-free software is always a challenging task. It gets even more challenging as the size of the project mushrooms and the time-to-market cycle gets shorter. It is the responsibility of every software developer/tester to ensure that the delivered software is of high quality and free from defects. Many software companies have deployed Defect Prevention (DP) as part of their software development process to reduce the huge cost that arises out of fixing defects. The purpose of DP is to identify the causes of defects and prevent them from recurring.

Identifying defects at the early stage of the software development process is one of the key objectives of DP. No doubt, one can achieve this by defining good unit and functional test cases and executing them to catch the defects before they reach the customer.

But how can one judge that the unit and functional test cases that the developer has written are good without having a proper tool to measure them? This is where code coverage tools like GNU 'gcov' and 'JCoverage/gpl' come into the picture. The purpose of code coverage tools is to tell you exactly which lines of your code have been executed and, most importantly, which aren't part of your unit and functional testing. In this article, I will demonstrate how these freely available tools measure and improve the effectiveness of the unit and functional test cases, and deliver a high quality defect free code.

Using GNU gcov

GNU gcov is part of the GNU GCC collection and can be used to get the code coverage data for programs written in C/C++. Figure 1 shows a simple C program that we will test and get code coverage with gcov.



```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void display1() {
5     printf("display1:\n");
6 }
7
8 void display2() {
9     printf("display2:\n");
10 }
11
12 void usage() {
13     printf("Usage: Test <[1-2]>\n");
14     exit(1);
15 }
16
17 int main(int argc, char *argv[]) {
18     int choice = 0;
19
20     if( argc != 2 ) {
21         usage();
22     }
23
24     choice = atoi(argv[1]);
25
26     if( choice == 1 ) {
27         display1();
28         exit(0);
29     }
30
31     if( choice == 2 ) display2(); else usage();
32
33     exit(0);
34 }

```

"test.c" 34L, 467C written

Figure 1: Program to be tested

To get the code coverage data with gcov, we need to compile the program with *-fprofile-arcs* and *-ftest-coverage* options. It should be noted that to get better results with gcov, no optimisation options (like *-O -O1 -O2* etc) should be used while compiling the program. Compiling with *-ftest-coverage* option will create two new files—'.bb' and '.bbg'. The '.bb' file contains a list of source files (including headers), functions within those files and line numbers corresponding to each basic block in the source file. The '.bbg' file is used to reconstruct the program flow graph for the

```

root@lvpc:/opt/lfy/codecoverage/c
# pwd
/opt/lfy/codecoverage/c
# ls
clean.sh run.sh test.c
# gcc -Wall -g -fprofile-arcs -ftest-coverage -o test test.c
# ls
clean.sh run.sh test test.bb test.bbg test.c
#

```

Figure 2: Compiling with code coverage options

source file. Figure 2 illustrates this.

The next step is to run/test the program to generate the 'profile' file (.da) as shown in Figure 3.

```

root@lvpc:/opt/lfy/codecoverage/c
# pwd
/opt/lfy/codecoverage/c
# ls
clean.sh run.sh test test.bb test.bbg test.c
# ./test 1
display1:
# ./test 2
display2:
#

```

Figure 3: Sample run

Once we generated the profile information, the next step is to get the code coverage data with gcov as shown in Figure 4.

```

root@lvpc:/opt/lfy/codecoverage/c
# pwd
/opt/lfy/codecoverage/c
# ls
clean.sh run.sh test test.bb test.bbg test.c test.da
# gcov -f test.c
100,00% of 2 source lines executed in function display1
100,00% of 2 source lines executed in function display2
0,00% of 3 source lines executed in function usage
90,00% of 10 source lines executed in function main
76,47% of 17 source lines executed in file test.c
Creating test.c.gcov.
#

```

Figure 4: Getting code coverage with gcov

Figure 5 clearly shows that our sample run (shown in Figure 3) covers only 76.47 per cent of the code. The lines starting with '#####' show that they have not been executed even once during our sample run. To get 100 per cent code

```

1      #include <stdio.h>
2      #include <stdlib.h>
3
4      1      void display1() {
5      1      printf("display1:\n");
6      }
7
8      1      void display2() {
9      1      printf("display2:\n");
10     }
11
12     ##### void usage() {
13     ##### printf("Usage: Test <[1-2]>\n");
14     ##### exit(1);
15     }
16
17     2      int main(int argc, char *argv[]) {
18     2      int choice = 0;
19
20     2      if( argc != 2 ) {
21     ##### usage();
22     }
23
24     2      choice = atoi(argv[1]);
25
26     2      if( choice == 1 ) {
27     1      display1();
28     1      exit(0);
29     }
30
31     1      if( choice == 2 ) display2(); else usage();
32
33     1      exit(0);
34     }
"test.c.gcov" 34L, 773C written

```

Figure 5: gcov output

```

root@lvpc:/opt/lfy/codecoverage/c
# pwd
/opt/lfy/codecoverage/c
# ls
clean.sh run.sh test test.bb test.bbg test.c
# ./test
Usage: Test <[1-2]>
# ./test 1
display1:
# ./test 2
display2:
# ./test 3
Usage: Test <[1-2]>
# ./test raja
Usage: Test <[1-2]>
#

```

Figure 6: Sample run with more test cases

coverage, we need to add few more test cases to our sample run as shown in Figure 6.

Running code coverage on the new profile data shows that 100 per cent of the code has been covered in the sample run (shown in Figure 6). Figures 7 and 8 illustrate this.

```

root@lvpc:/opt/lfy/codecoverage/c
# pwd
/opt/lfy/codecoverage/c
# ls
clean.sh run.sh test test.bb test.bbg test.c test.da
# gcov -f test.c
100,00% of 2 source lines executed in function display1
100,00% of 2 source lines executed in function display2
100,00% of 3 source lines executed in function usage
100,00% of 10 source lines executed in function main
100,00% of 17 source lines executed in file test.c
Creating test.c.gcov.
#

```

Figure 7: Getting code coverage with gcov

```

root@lvpc:/opt/lfy/codecoverage/c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  1 void display1() {
5  1   printf("display1:\n");
6  }
7
8  1 void display2() {
9  1   printf("display2:\n");
10 }
11
12 3 void usage() {
13 3   printf("Usage: Test <[1-2]>\n");
14 3   exit(1);
15 }
16
17 5 int main(int argc, char *argv[]) {
18 5   int choice = 0;
19
20 5   if( argc != 2 ) {
21 1   usage();
22 }
23
24 4   choice = atoi(argv[1]);
25
26 4   if( choice == 1 ) {
27 1   display1();
28 1   exit(0);
29 }
30
31 3   if( choice == 2 ) display2(); else usage();
32
33 1   exit(0);
34 }

"test.c.gcov" 34L, 773C written

```

Figure 8: gcov output

Using JCoverage/gpl

JCoverage/gpl is a free platform-independent tool that can be used to get code coverage data for programs written in Java. It adds instrumentation directly to the byte code to get the code

```

root@lvpc:/opt/lfy/codecoverage/java
1 public class Test {
2   public void display1() {
3     System.out.println("display1:");
4   }
5
6   public void display2() {
7     System.out.println("display2:");
8   }
9
10  public static void usage() {
11    System.out.println("Usage: Test <[1-2]>\n");
12    System.exit(1);
13  }
14
15  public static void main(String args[]) {
16    int choice = 0;
17
18    try {
19      Test test = new Test();
20
21      if( args.length == 0 ) {
22        Test.usage();
23      }
24
25      choice = Integer.parseInt(args[0]);
26
27      if( choice == 1 ) {
28        test.display1();
29        return;
30      }
31
32      if( choice == 2 ) test.display2(); else Test.usage();
33    } catch(NumberFormatException nfe) {
34      Test.usage();
35    } catch(Throwable t) {
36      t.printStackTrace();
37    }
38  }
39 }

"Test.java" 39L, 873C written

```

Figure 9: Program to be tested

coverage data. It can also be integrated with Ant to facilitate code coverage during the build process. You can get JCoverage/gpl from the website—<http://www.jcoverage.com>.

Figure 9 shows a simple Java program, that we will test and get code coverage with Jcoverage/gpl.

```

root@lvpc:/opt/lfy/codecoverage/java
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh run.sh Test.java
# javac Test.java
# ls
clean.sh run.sh Test.class Test.java
#

```

Figure 10: Compile source file

```

root@lvpc:/opt/lfy/codecoverage/java
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh run.sh Test.class Test.java
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.3/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar com.jcoverage.coverage.Instrument -d inst-classes Test.class
# ls
clean.sh inst-classes jcoverage.ser run.sh Test.class Test.java
# ls inst-classes/
Test.class
# ls -ltr jcoverage.ser
-rw-r--r-- 1 root root 1541 Sep 7 05:17 jcoverage.ser
#

```

Figure 11: Instrumenting with JCoverage

```

root@lvpc:/opt/lfy/codecoverage/java
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh inst-classes jcoverage.ser run.sh Test.class Test.java
# cp jcoverage.ser inst-classes/
# cd inst-classes/
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.3/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar Test 1
display1:
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.3/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar Test 2
display2:
# ls -ltr jcoverage.ser
-rw-r--r-- 1 root root 1938 Sep 7 05:18 jcoverage.ser
# ls -ltr ../jcoverage.ser
-rw-r--r-- 1 root root 1541 Sep 7 05:17 ../jcoverage.ser
# cp -f jcoverage.ser ../
cp: overwrite '../jcoverage.ser'? y
# cd ..
# pwd
/opt/lfy/codecoverage/java
#

```

Figure 12: Sample run



It should be noted that the JCoverage serialisation file (jcoverage.ser) should exist in the same directory as the instrumented class file. If not, you then need to explicitly copy the file to the instrumented class file location as shown in Figure 12.

```

root@lvpc:/opt/lfy/codecoverage/java
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh inst-classes jcoverage.ser run.sh Test.class Test.java
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.9/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar com.jcoverage.coverage.reporting.Main -o reports -s /opt/lfy/codecoverage/java -i jcoverage.ser
# ls
clean.sh inst-classes jcoverage.ser reports run.sh Test.class Test.java
# ls reports/
covr covd default.html images index.html Test.html
#

```

Figure 13: Generating JCoverage report

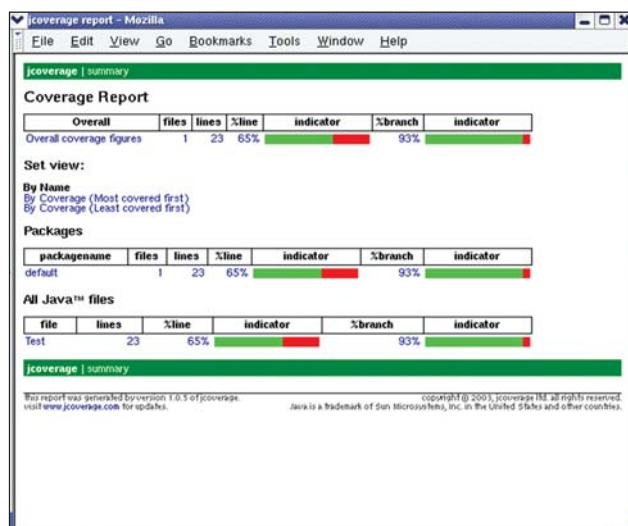


Figure 14: JCoverage summary report

Following are the steps to get code coverage with JCoverage /gpl.

Step 1: Compile the Java source file as shown in Figure 10.

Step 2: Instrument the class file with JCoverage as shown in Figure 11.

Step 3: Run/test the class (instrumented version) as shown in Figure 12.

Step 4: Generate the JCoverage report as shown in Figure 13.

Step 5: View code coverage report as shown in Figures 14 and 15.

Figure 15 clearly shows that our sample run (see Figure 12) does not cover 100 per cent of the code. The line that has hit count 0 indicates that it hasn't been executed even once during

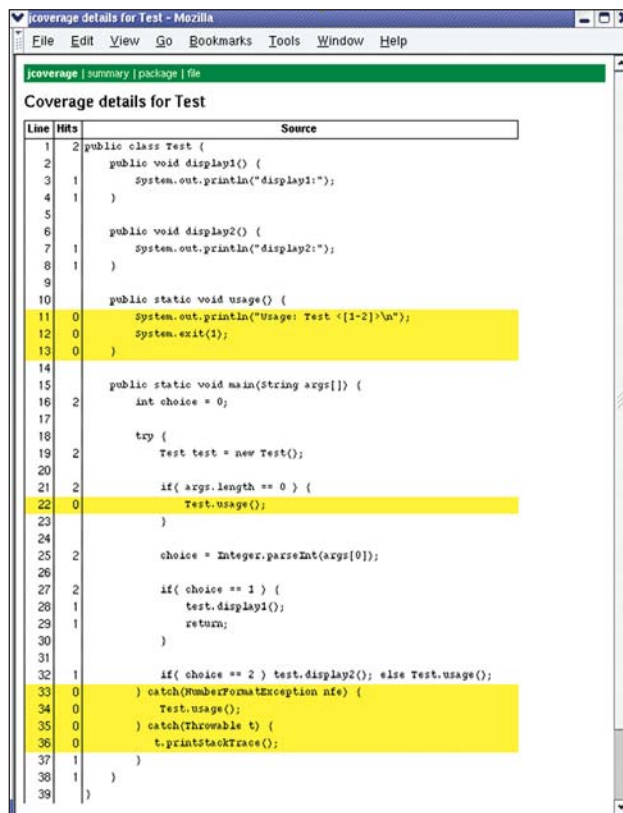


Figure 15: JCoverage detail report

```

root@lvpc:/opt/lfy/codecoverage/java
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh inst-classes jcoverage.ser run.sh Test.class Test.java
# cd inst-classes/
# pwd
/opt/lfy/codecoverage/java/inst-classes
# ls
jcoverage.ser Test.class
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.9/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar Test 1 display1
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.9/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar Test 2 display2
# java -classpath ./opt/jcoverage-1.0.5/jcoverage.jar:/opt/jcoverage-1.0.5/lib/bcel/5.1/bcel.jar:/opt/jcoverage-1.0.5/lib/gnu/getopt/1.0.9/java-getopt-1.0.9.jar:/opt/jcoverage-1.0.5/lib/junit/3.8.1/junit.jar:/opt/jcoverage-1.0.5/lib/log4j/1.2.8/log4j-1.2.8.jar:/opt/jcoverage-1.0.5/lib/oro/2.0.7/jakarta-oro-2.0.7.jar Test raja Usage: Test <[1-2]>
# cp jcoverage.ser ../
# cp: overwrite './jcoverage.ser'? y
# cd ..
# pwd
/opt/lfy/codecoverage/java
# ls
clean.sh inst-classes jcoverage.ser run.sh Test.class Test.java
#

```

Figure 16: Sample run (with added test cases)

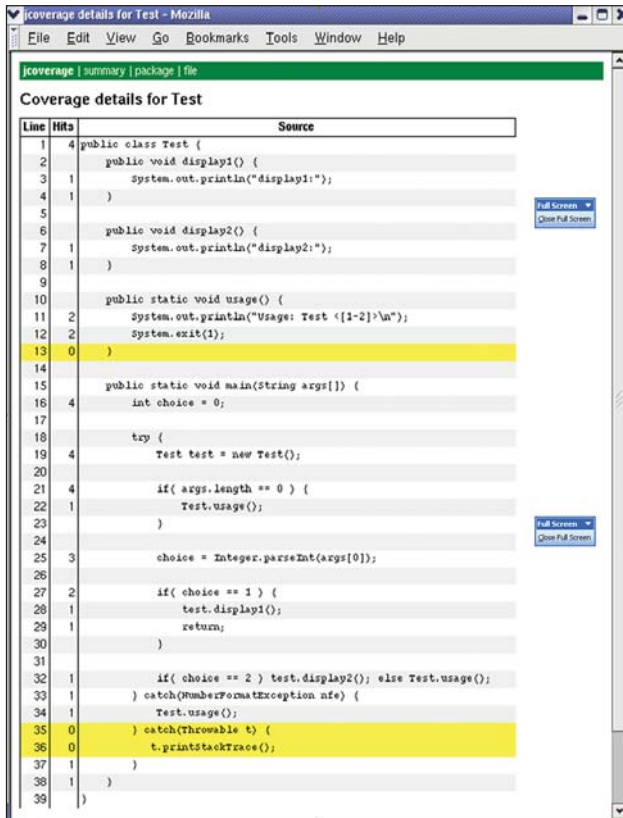


Figure 17: JCoverage detail report

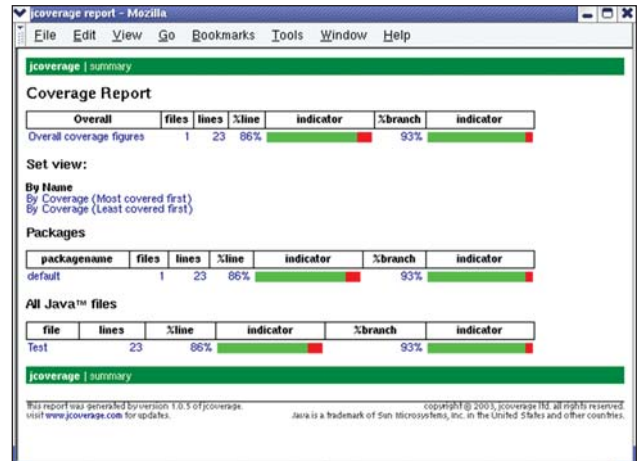


Figure 18: JCoverage summary report

the sample run. To get better code coverage we need to add a few more test cases to our sample run as shown in Figure 16.

Figures 17 and 18 show the coverage report.

Clearly, getting the code coverage data (using GNU gcov and JCoverage/gpl) to measure and improve the effectiveness of the unit/functional test cases, in order to deliver high quality defect-free code, is no longer a difficult task. **LFY**

By: Raja R K. The author is a lead engineer working with HCL Technologies (Cisco Systems Offshore Development Centre) in Chennai. He can be contacted at: rajark_hcl@yahoo.co.in

**We changed the careers of
600,000 Managers**

Are you one of them?

Register your profile on

naukri.com
India's No.1 Job Site

Log on to www.naukri.com today